



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

ΑΡΧΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Κεφάλαιο 18

Επιμέλεια:

Βασίλης Παλιουράς , Αναπληρωτής Καθηγητής
Ευάγγελος Δερματάς , Αναπληρωτής Καθηγητής
Σταύρος Νούσιας , Βοηθός Ερευνητή

Πολυτεχνική Σχολή

Τμήμα Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Υπολογιστών



Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο



ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου των διδασκόντων καθηγητών.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών» έχει χρηματοδοτηθεί μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Ανάπτυξη

Το παρόν εκπαιδευτικό υλικό αναπτύχθηκε στο τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών



Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης



- Αφαιρετικότητα στα δεδομένα
 - Έλεγχος τύπων
 - Δομές
 - Κατασκευή τύπων
- Δυναμική διαχείριση μνήμης
 - `free()`, `malloc()`, `realloc()`,...
- Αναδρομικός ορισμός συναρτήσεων



Αναδρομικότητα στη C

```
#include <stdio.h>
int f(int);

main ( ) {
    int a = 3;
    printf ("%d\n", f(a));
}

int f(int n) {
    int temp;
    if (n==0)
        temp = 1 ;
    else
        temp = n * f(n - 1);
    return temp;
}
```

Χρειάζεται συνθήκη τερματισμού.

Η συνάρτηση f() καλεί τον
εαυτό της στον **ορισμό** της.



Λύση χωρίς αναδρομικότητα

```
int f(int n) {  
    int temp = 1 , i;  
  
    for (i=1;i<=n;i++)  
        temp *= i;  
  
    return temp;  
}
```

```
int f(int n) {  
    int temp = 1 , i;  
  
    for(i=1;i<=n;temp*=(i+  
+));  
  
    return temp;  
}
```



Ένας ορισμός της countchar()

```
int countchar(char *s, char c) {  
    return (*s)?(*s == c) + countchar(s+1,c):0;  
}
```



Ένας ορισμός της getmin()

```
int getmin (int a[], int size) {  
    int i , temp;  
    temp = a[0];  
  
    for (i=1; i < size; i++) {  
        if ( temp>a[i] ) temp = a[i];  
    }  
  
    return temp;  
}
```



Ένας ορισμός της getMax()

```
int getMax (int a[], int size) {  
    int i , temp;  
    temp = a[0];  
  
    for (i=1; i < size; i++) {  
        if ( temp<a[i] ) temp = a[i];  
    }  
  
    return temp;  
}
```



getmin() & getmax()

```
int getmin (int a[], int
size) {
    int i , temp;
    temp = a[0];

    for (i=1; i < size; i++) {
        if ( temp>a[i] ) temp =
        a[i];
    }

    return temp;
}
```

```
int getmax (int a[], int
size) {
    int i , temp;
    temp = a[0];

    for (i=1; i < size; i++) {
        if ( temp<a[i] ) temp =
a[i];
    }

    return temp;
}
```



Όνομα συνάρτησης

- Οι παρενθέσεις ως τελεστής
 - Δήλωση/πρότυπο συνάρτησης
int f (int);
 - Κλήση συνάρτησης
int a ;
a = f(5) ;
- Το όνομα συνάρτησης μόνο του \Rightarrow διεύθυνση
- Μπορώ να δηλώσω σχετική **μεταβλητή**



Δείκτης σε συνάρτηση

```
/* δήλωση δείκτη σε ακέραιο */
```

```
int *intptr;
```

```
/* πρότυπο συνάρτησης που επιστρέφει δείκτη σε ακέραιο */
```

```
int *f();
```

```
/* δήλωση δείκτη σε συνάρτηση */
```

```
int (*f)();
```

```
int function1(void);
```

```
int function2(void);
```

```
main ( ) {
```

```
int a = 0;
```

```
int (*f)( );
```

```
...
```

```
if (!a)
```

```
    f = function1;
```

```
else
```

```
    f= function2;
```

```
(*f)();
```

```
}
```



Παράδειγμα

```
#include <stdio.h>

int map (int a[], int size, int (*f)(int,
int));
int more(int a, int b);
int less(int a, int b);

int main(void) {

int data[7]= {-1, 2, 1, 0, 5, 7, -3};

printf("max: %2d\n", map(data, 7,
more));
printf("max: %2d\n", map(data, 7,
less));

return 0;
}

int less (int a, int b) {
return a > b;
}

int more (int a, int b) {
return a < b;
}

int map (int a[], int size, int (*f)(int,
int)) {
int i , temp;
temp = a[0];

for (i=1; i < size; i++)
if ( f(temp, a[i])) temp = a[i];

return temp;
}
```



Πίνακας Δεικτών σε Συνάρτηση - Αρχικοποίηση

```
#include <stdio.h>

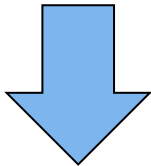
double one( double );
double two( double );
main ( ) {
double (*f[2])(double) = {one, two} ;
double x = 3.47, y;
int i ;
    do {
        printf("select function:\t");
        scanf("%d", &i);
        y = (*f[i-1])( x) ;
        printf("result %g\n", y);
    } while (1) ;
}
double one(double x) { return (x + 1.0) ; }
double two(double x ) { return (x + 2.0) ; }
```



Λίστα

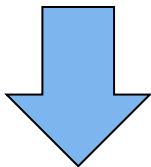
```
Listelement c = { "Giannis", 5, NULL};
```

Στατική δήλωση



Δυναμική δήλωση στοιχείου της λίστας

```
Listelement *element_ptr;  
element_ptr = (Listelement *) malloc ( sizeof (Listelement));  
strcpy((*element_ptr).name, "Ntina");  
(*element_ptr).age = 9;  
(*element_ptr).next = NULL;
```



Άλλος συμβολισμός, ενδεικτικά:
element_ptr->name αντί (*element_ptr).name

```
Listelement *element_ptr;  
element_ptr = (Listelement *) malloc ( sizeof (Listelement));  
strcpy(element_ptr->name, "Ntina");  
element_ptr->age = 9;  
element_ptr->next = NULL;
```


Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define L 30
struct listelement {
    char name[L];
    int age;
    struct listelement *next;
};
typedef struct listelement Listelement;
typedef Listelement * Listelement_ptr;
int main(int argc, char *argv[]) {
    /* Initialize */
    Listelement_ptr iterator=NULL, element_ptr, previous_ptr, first_ptr; /* create elements and put them to list*/
    element_ptr = (Listelement_ptr) malloc ( sizeof (Listelement));
    strcpy(element_ptr->name,"Ntina");
    element_ptr -> age = 9;
    element_ptr -> next = NULL;
    first_ptr = element_ptr;
    previous_ptr = element_ptr;
    element_ptr = (Listelement_ptr) malloc ( sizeof (Listelement));
    strcpy(element_ptr->name,"Giannis");
    element_ptr -> age = 6 ;
    previous_ptr -> next = element_ptr;
    /* Access elements */
    for (iterator = first_ptr; iterator != NULL ; iterator = iterator->next) {
        printf( "name: %s\n", iterator->name);
        printf("age: %d\n", iterator->age);
    }
    return 0;
}
```



mytypes.h

```
#define L 30
struct listelement {
    char name[L];
    int age;
    struct listelement *next;
};
typedef struct listelement Listelement;
typedef Listelement * Listelement_ptr;
typedef Listelement * List;
```



Τύποι λίστας στο `mytypes.h`

- `Listelement`
 - Στοιχείο λίστας (`struct listelement`)
- `Listelement_ptr`
 - Δείκτης σε στοιχείο λίστας. Ίδιο με
 - `struct listelement *`
 - `Listelement *`
- `List`
 - Δείκτης σε στοιχείο λίστας
 - Μεταβλητές αυτού του τύπου δείχνουν
 - Στο πρώτο στοιχείο της λίστας ή
 - Έχουν την τιμή `NULL` (κενή λίστα).



Δημιουργία στοιχείου

```
#include <string.h>
#include <stdlib.h>
#include "mytypes.h"
```

```
Listelement_ptr createnewelement(char word[], int number) {
    Listelement_ptr newelement_ptr;

    newelement_ptr = (Listelement_ptr) malloc ( sizeof (Listelement));
    strcpy(newelement_ptr->name, word);
    newelement_ptr -> age = number;
    newelement_ptr -> next = NULL;

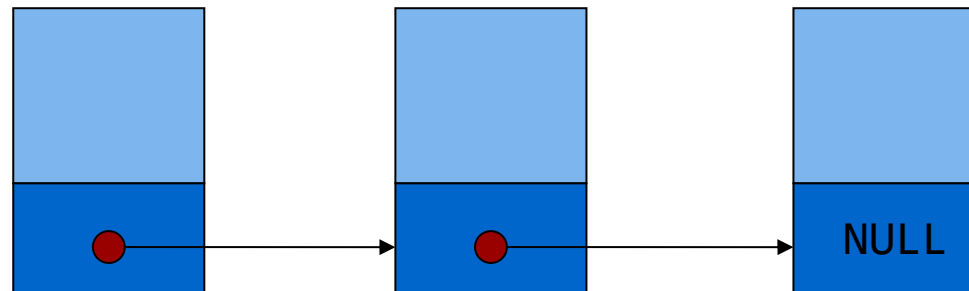
    return newelement_ptr;
}
```



Διατρέχει όλα τα στοιχεία της λίστας

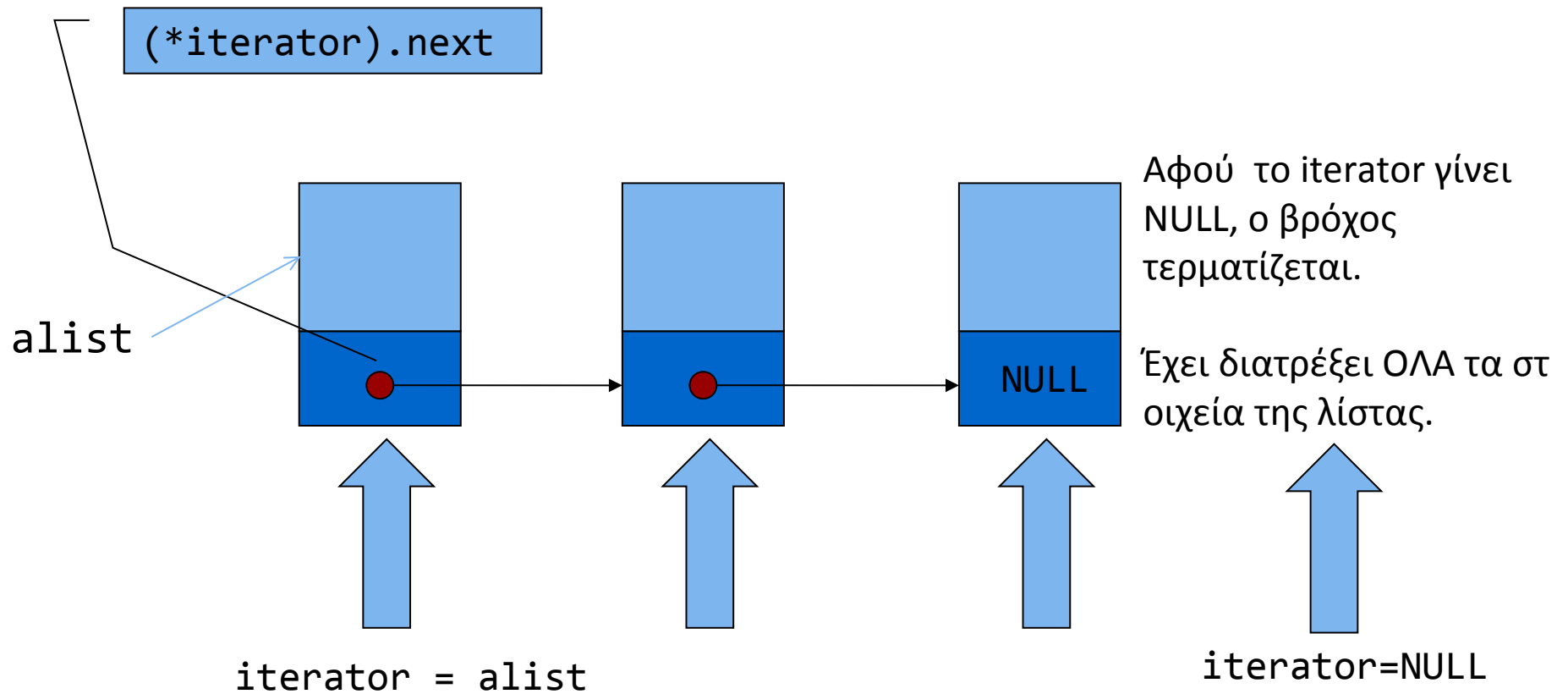
```
#include <stdio.h>
#include "mytypes.h"

void reportlist(List const alist) {
    Listelement_ptr iterator= alist;
    for (; iterator != NULL ; iterator = iterator->next)
    {
        printf( "name: %s\n", iterator->name);
        printf("age: %d\n", iterator->age);
    }
}
```



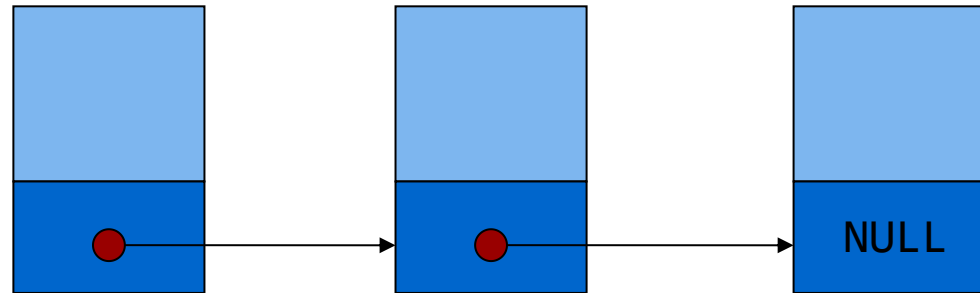
Δημιουργία στοιχείου

```
Listelement_ptr iterator = alist;
```

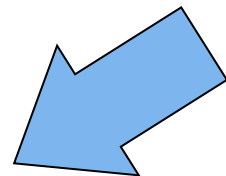


```
for (; iterator != NULL ; iterator = iterator->next)
```

Προσθήκη στοιχείου στο τέλος λίστας



```
#include <stdio.h>  
#include "mytypes.h"
```



Παράμετρος δείκτης σε λίστα ή διπλός δείκτης σε
στοιχείο

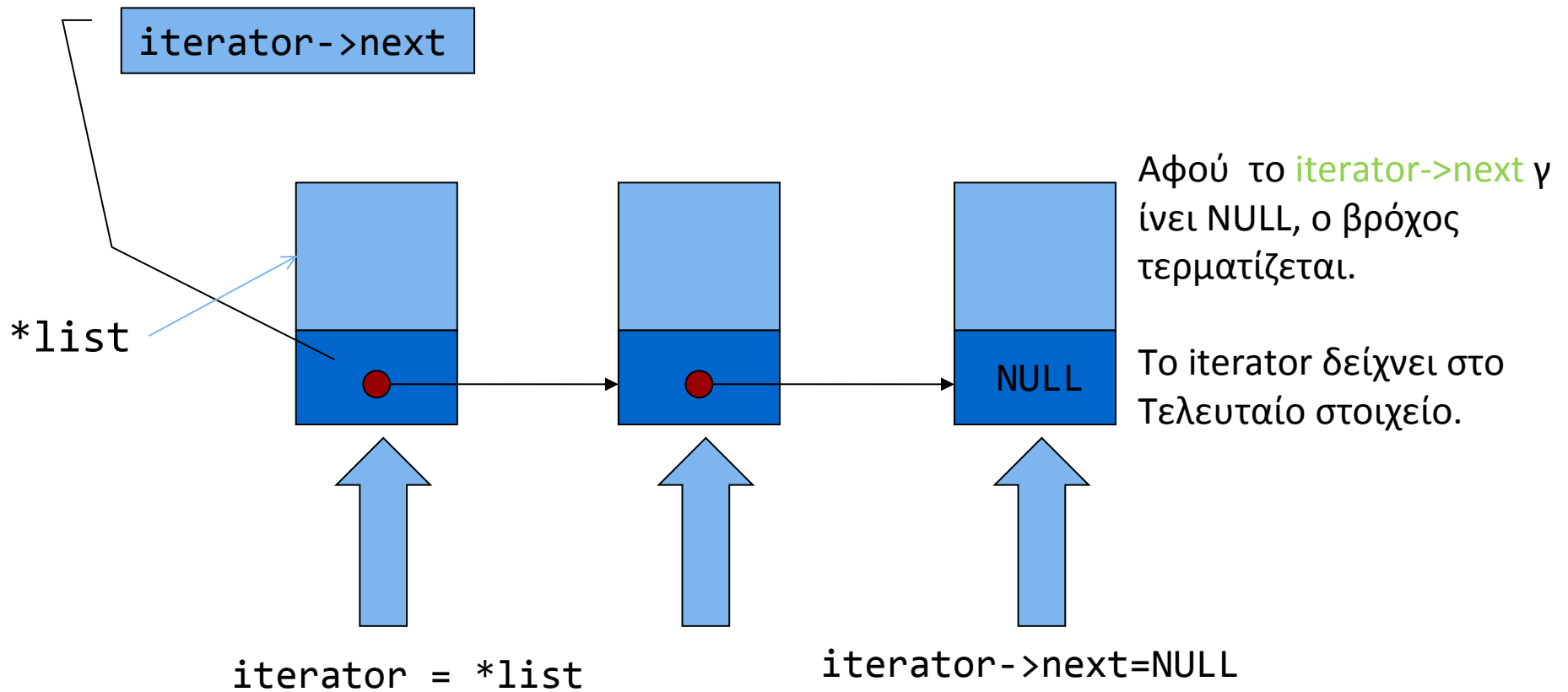
```
void addtolist (List *list, Listelement_ptr elem_ptr ) {  
    Listelement_ptr iterator = *list;  
    if (*list==NULL) /* handle empty list*/  
        *list = elem_ptr;  
    else { /* if not empty, move to last element list*/  
        for (; iterator->next != NULL; iterator = iterator->next);  
        /* append element to list */  
        iterator->next = elem_ptr;  
    }  
}
```



Όταν βγει από το βρόχο, ο iterator δείχνει στο τελευταίο στοιχείο

iterator->next

```
Listelement_ptr iterator = *list;
```



```
for (; iterator->next != NULL ; iterator = iterator->next)
```



Παράδειγμα χρήσης συναρτήσεων

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mytypes.h"
Listelement_ptr createnewelement(char *, int);
Listelement_ptr findelementbyname(List, char []);
void reportlist( List const);
void addtolist(List *, Listelement_ptr);
void deleteelement(List *, Listelement_ptr);
int main(int argc, char *argv[]) {
    /* Initialize */
    Listelement_ptr iterator=NULL, element_ptr, previous_ptr;
    List nameslist = NULL;
    char name[50];
    /* create elements and put them to list*/
    element_ptr = createnewelement("Ntina", 10);
    addtolist (&nameslist, element_ptr);
    element_ptr = createnewelement("Giannis", 5);
    addtolist( &nameslist, element_ptr);
    element_ptr = createnewelement("Maria", 7);
    addtolist( &nameslist, element_ptr);
    reportlist(nameslist);
    /* find, delete, report */
    do {
        printf("name: ");
        scanf("%s", name);
        element_ptr = findelementbyname(nameslist, name);
        if (element_ptr) {
            printf("found it. data: %d\n", element_ptr->age);
            deleteelement(&nameslist, element_ptr);
            reportlist(nameslist);
        }
    } while (nameslist!=NULL);
    return 0;
}
```



Λίστα ως παράμετρος με αναφορά

```
void addtolist (List *, Listelement_ptr );
```

Η nameslist αποθηκεύεται στη θέση 22FE48.

Η τιμή της nameslist αλλάζει μετά την κλήση της addtolist.

(όχι η θέση της!)

Τύπος της nameslist: List Τύπος της θέσης της: List *

Τιμές **πριν** την
εκτέλεση της addtolist



Τιμές **μετά** την
εκτέλεση της addtolist



```
C:\Program Files (x86)\Dev-Cpp\ConsolePaus
nameslist: 0
&nameslist: 22FE48
element_ptr: 2F7BB0
Press any key to continue
nameslist: 2F7BB0
&nameslist: 22FE48
element_ptr: 2F7BB0
Press any key to continue
```

```
List nameslist = NULL;
```

```
Listelement_ptr element_ptr;
```

```
element_ptr = createnewelement("Ntina", 10);
```

```
addtolist (&nameslist, element_ptr);
```



Λίστα ως παράμετρος με αναφορά

Θέση της namelist (ή &nameslist)

22FE48

Τιμή της namelist

2F7BB0

Πριν την κλήση της `addtolist`

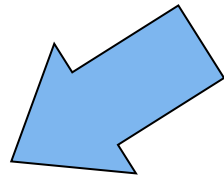
Μετά την κλήση της `addtolist`

```
C:\Program Files (x86)\Dev-Cpp\ConsolePaus
nameslist:      0
&nameslist:    22FE48
element_ptr:   2F7BB0
Press any key to continue
nameslist:    2F7BB0
&nameslist:  22FE48
element_ptr:  2F7BB0
Press any key to continue
```



Διαγραφή στοιχείου

```
#include <stdio.h>
#include <stdlib.h>
#include "mytypes.h"
```



Παράμετρος: δείκτης σε λίστα ή
διπλός δείκτης σε στοιχείο λίστας

```
int deleteelement(List *alist, Listelement_ptr element) {
    Listelement_ptr iterator = *alist, todelete_ptr;
    if (element == NULL || *alist == NULL)
        return -1 ;
    if (element== *alist) { /* if required, delete first element */
        *alist = element->next ;
        free(element);
    }
    else { /* find the element before the one to be deleted */
        for(; iterator->next != element; iterator = iterator->next) ;
        iterator->next = element->next;
        free(element);
    }
    return 0;
}
```



Διαγραφή με πρόβλεψη το στοιχείο να μην υπάρχει στη λίστα

```
#include <stdio.h>
#include <stdlib.h>
#include "mytypes.h"
int deleteelement(List *alist, Listelement_ptr todelete_ptr) {
    Listelement_ptr iterator = *alist;
    if (todelete_ptr == NULL || *alist == NULL)
        return -1 ;
    if (todelete_ptr == *alist) {
        *alist = todelete_ptr->next ;
        free(todelete_ptr);
    }
    else {
        for(; iterator != NULL && iterator->next != todelete_ptr ;
            iterator = iterator->next) ;
        if (iterator!=NULL) {
            iterator->next = todelete_ptr->next;
            free(todelete_ptr);}
        else {
            printf("element not in list");
            return -1;
        }
    }
    return 0;
}
```

Όσο δείχνεις σε στοιχείο της λίστας (iterator!=NULL) ΚΑΙ το επόμενο στοιχείο δεν είναι το προς διαγραφή (iterator->next != todelete_ptr), πήγαινε στο επόμενο στοιχείο

Αν μετά το βρόχο το iterator δείχνει σε στοιχείο (!=NULL), αυτό είναι το ακριβώς προηγούμενο από το προς διαγραφή.



Διαγραφή με πρόβλεψη το στοιχείο να μην υπάρχει στη λίστα

```
#include <stdio.h>
#include <stdlib.h>
#include "mytypes.h"
int deleteelement(List *alist, Listelement_ptr todelete_ptr) {
    Listelement_ptr iterator = *alist;
```

```
    if (todelete_ptr == NULL || *alist == NULL)
        return -1 ;
```

```
    if (todelete_ptr == *alist) { /* if required, delete first element*/
        *alist = todelete_ptr->next ;
        free(todelete_ptr);
    }
```

Πρώτα χρησιμοποιώ το todelete_ptr
Μετά αποδεσμεύεται με free

```
    else { /* find the element before the one to be deleted or
            continue until no more elements in list */
```

```
        for(; iterator != NULL && iterator->next != todelete_ptr ;
            iterator = iterator->next) ;
```

```
        if (iterator!=NULL) {
            iterator->next = todelete_ptr->next;
            free(todelete_ptr);}
```

```
        else {
            printf("element not in list");
            return -1;
        }
```

Τι θα γίνει αν γράψουμε
iterator->next!=todelete_ptr && iterator !=NULL
αντί για
iterator!=NULL && iterator->next!=todelete_ptr

```
    }
    return 0;
```



Χρήσιμα σημεία σε συναρτήσεις επεξεργασίας λίστας

- Η λίστα ως παράμετρος
 - Αναφερόμαστε σε λίστα με τη διεύθυνση του πρώτου στοιχείου
 - Μερικές συναρτήσεις αλλάζουν τη διεύθυνση του πρώτου στοιχείου (προσθέτουν, διαγράφουν, ...)
 - Άλλες όχι (εκτύπωση, καταμέτρηση, αναζήτηση...)
- Πρώτα χρησιμοποιώ, μετά διαγράφω
- Βρίσκω και καλύπτω ειδικές περιπτώσεις
 - Ενδεικτικά: άδεια λίστα, πρώτο στοιχείο, κενό στοιχείο ως είσοδος



Αναδρομική αναζήτηση σε λίστα

```
#include <string.h>
#include "mytypes.h"

Listelement_ptr recfind(List alist, char name[]) {
    if (alist == NULL)
        return NULL;
    else
        if (!strcmp(alist->name, name))
            return alist;
        else
            return recfind(alist->next, name);
}
```



Χρήση αρχείων

```
#include <string.h>
#include "mytypes.h"

Listelement_ptr recfind(List alist, char name[]) {
    if (alist == NULL)
        return NULL;
    else
        if (!strcmp(alist->name, name))
            return alist;
        else
            return recfind(alist->next, name);
}
```



Χρήση αρχείων

```
#include <stdio.h>
#include <stdlib.h>

main ( ) {
    int i;
    int num;
    FILE *data ;

    if ((data = fopen ("dataout.txt", "wt") )==NULL) {
        printf("Cannot create file.\n");
        exit(1);
    }
    for ( i = 0; i < 10 ; i++)
        fprintf (data, "number i: %d\n", i);
    fclose(data);

    if((data = fopen ("numbers.txt", "rt"))==NULL) {
        printf("Cannot read from file.\n");
        exit(1);
    };
    while (fscanf(data, "%d", &num)!=EOF)
        printf("%d ", num);
    fclose(data);
}
```



Χρήση αρχείων

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int n;
    FILE * pFile;
    char buffer [27];

    pFile = fopen ("test.txt","w+");

    for ( n='A' ; n<='Z' ; n++)
        fputc ( n, pFile);

    rewind (pFile);
    fread (buffer,1,26,pFile);
    fclose (pFile);
    buffer[26]='\0';
    puts (buffer);

    system("PAUSE");
    return 0;
}
```



fwrite()

```
size_t fwrite ( const void * ptr, size_t size, size_t count, FILE *stream );
```

- Γράφει περιοχή μνήμης στη ροή stream
- Παράμετροι
 - ptr Δείκτης σε πίνακα στοιχείων προς εγγραφή.
 - size μέγεθος κάθε στοιχείου σε bytes
 - count πλήθος στοιχείων καθένα μεγέθους *size* bytes.
 - stream Δείκτης σε FILE



Χρήση αρχείων

```
size_t fread ( void * ptr, size_t size, size_t count, FILE * stream );
```

- Παράμετροι
 - Ptr Δείκτης σε μπλοκ μνήμης ελάχιστου μεγέθους (*size*count*) bytes.
 - Size Μέγεθος σε bytes κάθε στοιχείου προς ανάγνωση.
 - Count Πλήθος στοιχείων καθένα μεγέθους *size* bytes.
 - Stream Δείκτης σε [FILE](#)



Χρήση αρχείων

```
#include <stdio.h>
#include <stdlib.h>

typedef struct data{
    int a;
    char name[10];
} Data;

int main(int argc, char *argv[]) {

    Data mydata[] = {{5, "red"}, {10, "black"}, {11,"yellow"}};
    Data mycopy[10];
    FILE *testio;
    int i;

    testio = fopen ("mydata.bin", "wb");
    fwrite (mydata, sizeof (Data), 3, testio);
    fclose(testio);

    testio = fopen ("mydata.bin", "rb");
    fread(mycopy, sizeof (Data), 3, testio);
    fclose(testio);

    for (i=0;i<3;i++)
        printf("value: %d\nname: %s\n", mycopy[i].a, mycopy[i].name);
    return 0;
}
```



Χρήση αρχείων

- `int fseek (FILE * stream, long int offset, int origin);`
- Παράμετροι
 - Δείκτης σε `FILE` .
 - Offset πλήθος bytes για πρόσθεση στο *origin*.
 - Origin τρέχουσα θέση στην οποία προστίθεται το *offset*.
 - Μπορεί να είναι μια από τις σταθερές:
 - `SEEK_SET` Αρχή αρχείου
 - `SEEK_CUR` Τρέχουσα θέση
 - `SEEK_END` Τέλος αρχείου



Χρήση αρχείων

```
#include <stdio.h>
#include <stdlib.h>

typedef struct data{
    int a;
    char name[10];
} Data;

int main(int argc, char *argv[])
{
    FILE * testio;
    long filesize;
    Data * mycopy;
    size_t result;
    int numberofelems;
    int i;

    testio = fopen ( "mydata.bin" , "rb" );
    if (testio == NULL)
        printf("failed to open\n");
    else
        printf("stream open\n");

    fseek (testio, 0 , SEEK_END);
    filesize = ftell (testio);
    rewind (testio);
```

```
mycopy = (Data *) malloc (sizeof (char)*filesize);

    result = fread (mycopy,1,filesize,testio);
    if (result != filesize) {fputs ("Reading error",stderr);
        exit (3);}

    fclose (testio);

    numberofelems=(filesize)/(sizeof(Data));
    printf("Data elements read: %d\n", numberofelems);

    for (i=0;i<numberofelems;i++)
        printf("value: %d\nname: %s\n", mycopy[i].a,
            mycopy[i].name);

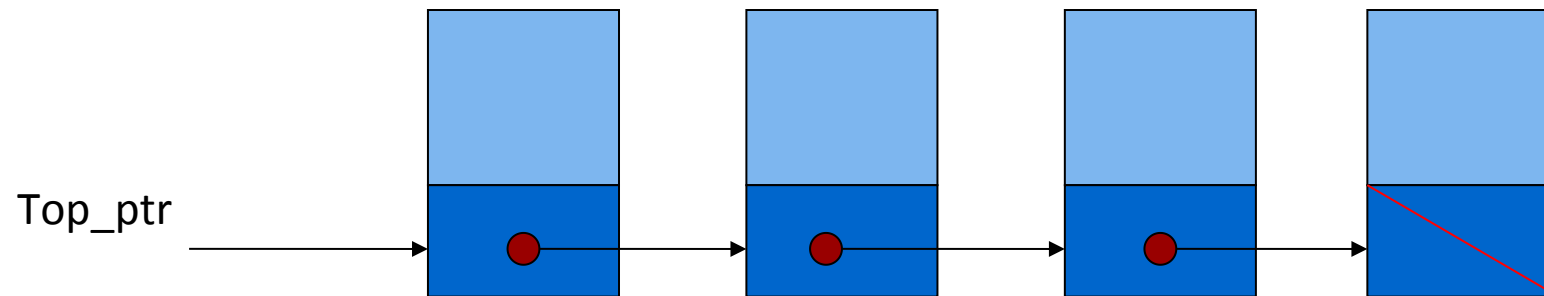
    free (mycopy);

    system("PAUSE");
    return 0;
}
```



Στοιίβες - stacks

- Λειτουργίες push/pop



Χρήση αρχείων

```
#include <stdio.h>
#include <stdlib.h>

struct stackNode { /* self-referential structure */
    int data;
    struct stackNode *nextPtr;
};

typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;

void push( StackNodePtr *, int );
int pop( StackNodePtr * );
int isEmpty( StackNodePtr );
void printStack( StackNodePtr );
void instructions( void );
```

```
int main()
{
    StackNodePtr stackPtr = NULL; /* points to stack top */
    int choice, value;

    instructions();
    printf( "? " );
    scanf( "%d", &choice );

    while ( choice != 3 ) {

        switch ( choice ) {
            case 1: /* push value onto stack */
                printf( "Enter an integer: " );
                scanf( "%d", &value );
                push( &stackPtr, value );
                printStack( stackPtr );
                break;
            case 2: /* pop value off stack */
                if ( !isEmpty( stackPtr ) )
                    printf( "The popped value is %d.\n",
                        pop( &stackPtr ) );

                printStack( stackPtr );
                break;
            default:
                printf( "Invalid choice.\n\n" );
                instructions();
                break;
        }

        printf( "? " );
        scanf( "%d", &choice );
    }

    printf( "End of run.\n" );
    return 0;
}
```



Χρήση αρχείων

```
/* Print the instructions */
void instructions( void )
{
    printf( "Enter choice:\n"
           "1 to push a value on the stack\n"
           "2 to pop a value off the stack\n"
           "3 to end program\n" );
}

/* Insert a node at the stack top */
void push( StackNodePtr *topPtr, int info )
{
    StackNodePtr newPtr;

    newPtr = malloc( sizeof( StackNode ) );
    if ( newPtr != NULL ) {
        newPtr->data = info;
        newPtr->nextPtr = *topPtr;
        *topPtr = newPtr;
    }
    else
        printf( "%d not inserted. No memory available.\n",
              info );
}
```

```
/* Print the stack */
void printStack( StackNodePtr currentPtr )
{
    if ( currentPtr == NULL )
        printf( "The stack is empty.\n\n" );
    else {
        printf( "The stack is:\n" );

        while ( currentPtr != NULL ) {
            printf( "%d --> ", currentPtr->data );
            currentPtr = currentPtr->nextPtr;
        }

        printf( "NULL\n\n" );
    }
}

/* Is the stack empty? */
int isEmpty( StackNodePtr topPtr )
{
    return topPtr == NULL;
}

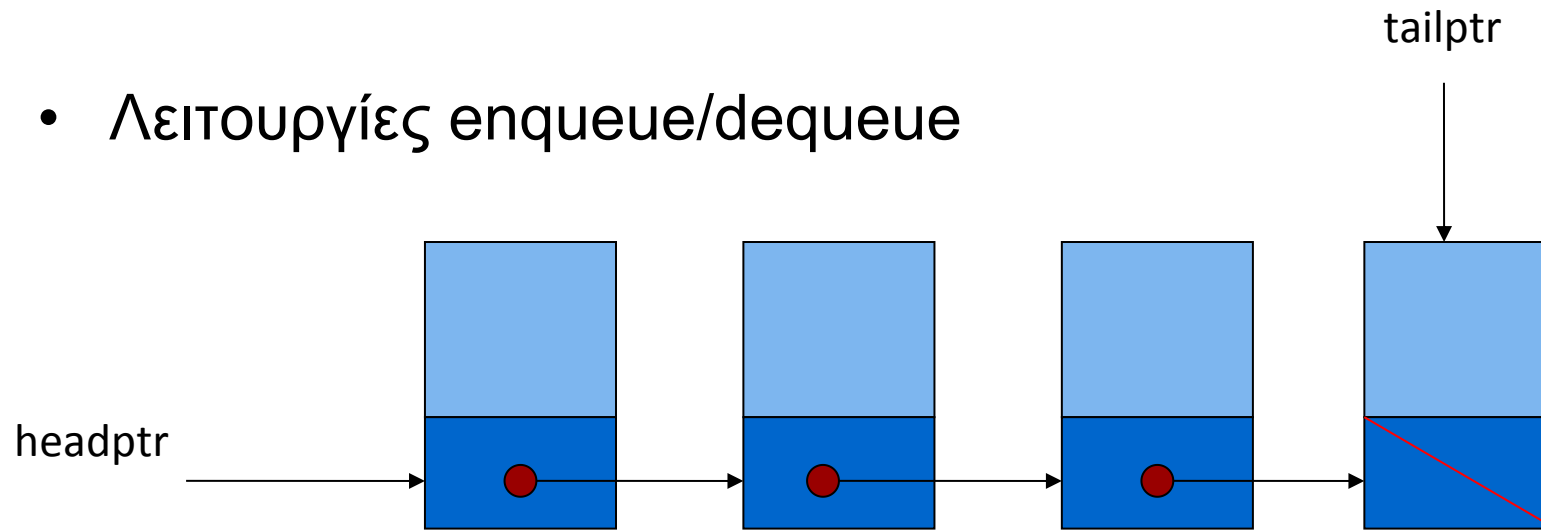
/* Remove a node from the stack top */
int pop( StackNodePtr *topPtr )
{
    StackNodePtr tempPtr;
    int popValue;

    tempPtr = *topPtr;
    popValue = ( *topPtr )->data;
    *topPtr = ( *topPtr )->nextPtr;
    free( tempPtr );
    return popValue;
}
```



Ουρές - Queues

- Λειτουργίες enqueue/dequeue



Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>

struct queueNode { /* self-referential structure */
    char data;
    struct queueNode *nextPtr;
};

typedef struct queueNode QueueNode;
typedef QueueNode *QueueNodePtr;

/* function prototypes */
void printQueue( QueueNodePtr );
int isEmpty( QueueNodePtr );
char dequeue( QueueNodePtr *, QueueNodePtr * );
void enqueue( QueueNodePtr *, QueueNodePtr *, char );
void instructions( void );
```

```
void instructions( void )
{
    printf ( "Enter your
             choice:\n"
            " 1 to add an item
             to the queue\n"
            " 2 to remove an
             item from the queue\n"
            " 3 to end\n" );
}
```



Παράδειγμα

```
int main()
{
    QueueNodePtr headPtr = NULL,
                  tailPtr = NULL;

    int choice;
    char item;

    instructions();
    printf( "? " );
    scanf( "%d", &choice );

    while ( choice != 3 ) {
        switch( choice ) {
            case 1:
                printf( "Enter a character: " );
                scanf( "\n%c", &item );
                enqueue( &headPtr, &tailPtr,
                        printQueue( headPtr );
                break;
```

```
            case 2:
                if ( !isEmpty( headPtr ) ) {
                    item = dequeue( &headPtr, &tailPtr );
                    printf( "%c has been dequeued.\n", item );
                }

                printQueue( headPtr );
                break;

            default:
                printf( "Invalid choice.\n\n" );
                instructions();
                break;
        }

        printf( "? " );
        scanf( "%d", &choice );
    }

    printf( "End of run.\n" );
    return 0;
}
```



Παράδειγμα

```
void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr,
             char value )
{
    QueueNodePtr newPtr;

    newPtr = malloc( sizeof( QueueNode ) );

    if ( newPtr != NULL ) {
        newPtr->data = value;
        newPtr->nextPtr = NULL;

        if ( isEmpty( *headPtr ) )
            *headPtr = newPtr;
        else
            ( *tailPtr )->nextPtr = newPtr;

        *tailPtr = newPtr;
    }
    else
        printf( "%c not inserted. No memory available.\n",
              value );
}
```

```
char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr )
{
    char value;
    QueueNodePtr tempPtr;

    value = ( *headPtr )->data;
    tempPtr = *headPtr;
    *headPtr = ( *headPtr )->nextPtr;

    if ( *headPtr == NULL )
        *tailPtr = NULL;

    free( tempPtr );
    return value;
}

int isEmpty( QueueNodePtr headPtr )
{
    return headPtr == NULL;
}

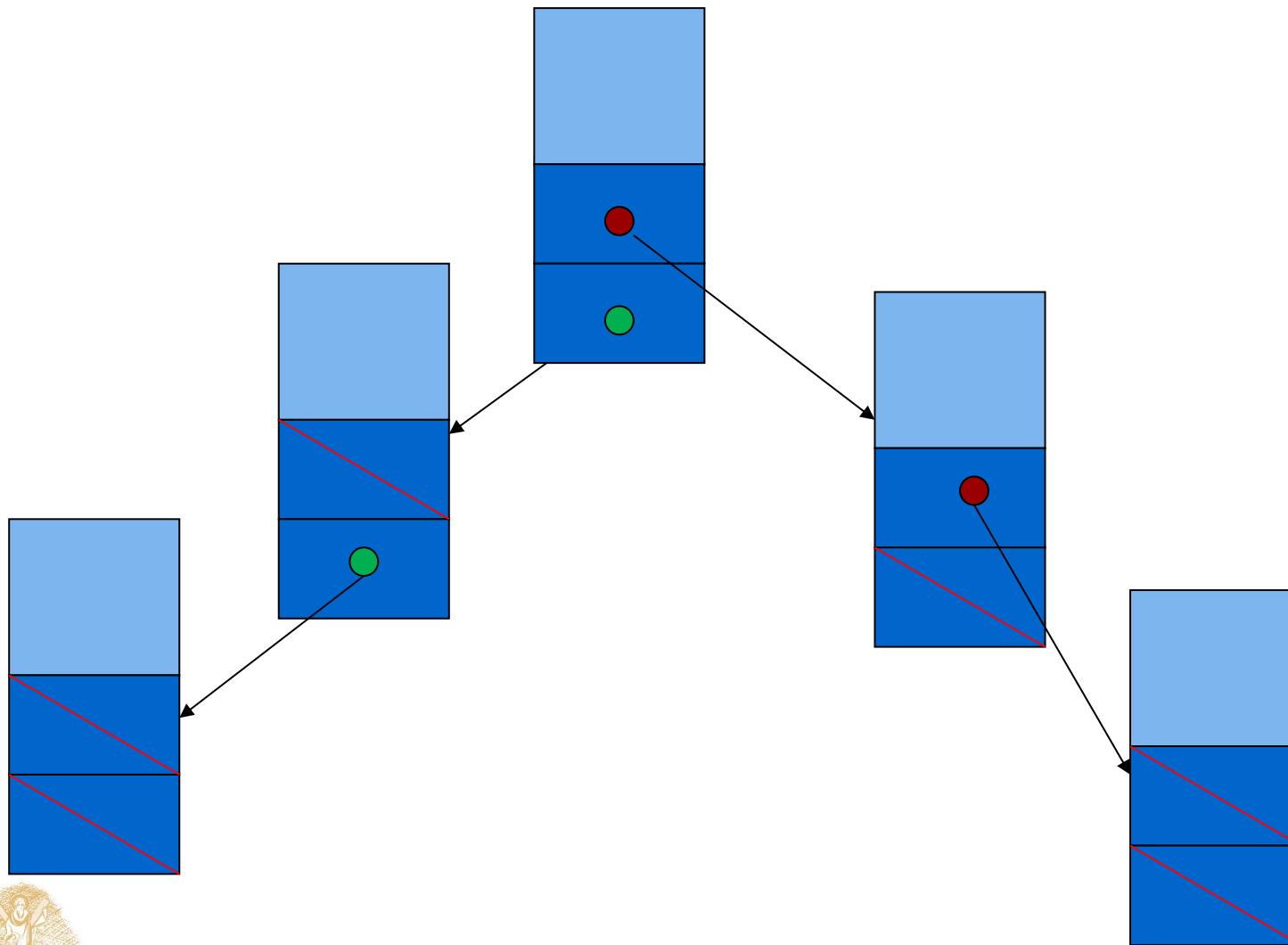
void printQueue( QueueNodePtr currentPtr )
{
    if ( currentPtr == NULL )
        printf( "Queue is empty.\n\n" );
    else {
        printf( "The queue is:\n" );

        while ( currentPtr != NULL ) {
            printf( "%c --> ", currentPtr->data );
            currentPtr = currentPtr->nextPtr;
        }

        printf( "NULL\n\n" );
    }
}
```



Διαδικά δέντρα



Δυαδικά Δένδρα

- Πρακτικές αναδρομικές συναρτήσεις
 - για κατασκευή του δυαδικού δένδρου και
 - για αναζήτηση



Διαδικά Δένδρα

- Γραμμική δυαδική αναζήτηση σε πίνακα
 - Βρες αν υπάρχει το 5



- Πρέπει να είναι ταξινομημένος ο πίνακας
- Πώς προσθέτω στοιχεία σε ταξινομημένο πίνακα;



Δυαδικά Δένδρα

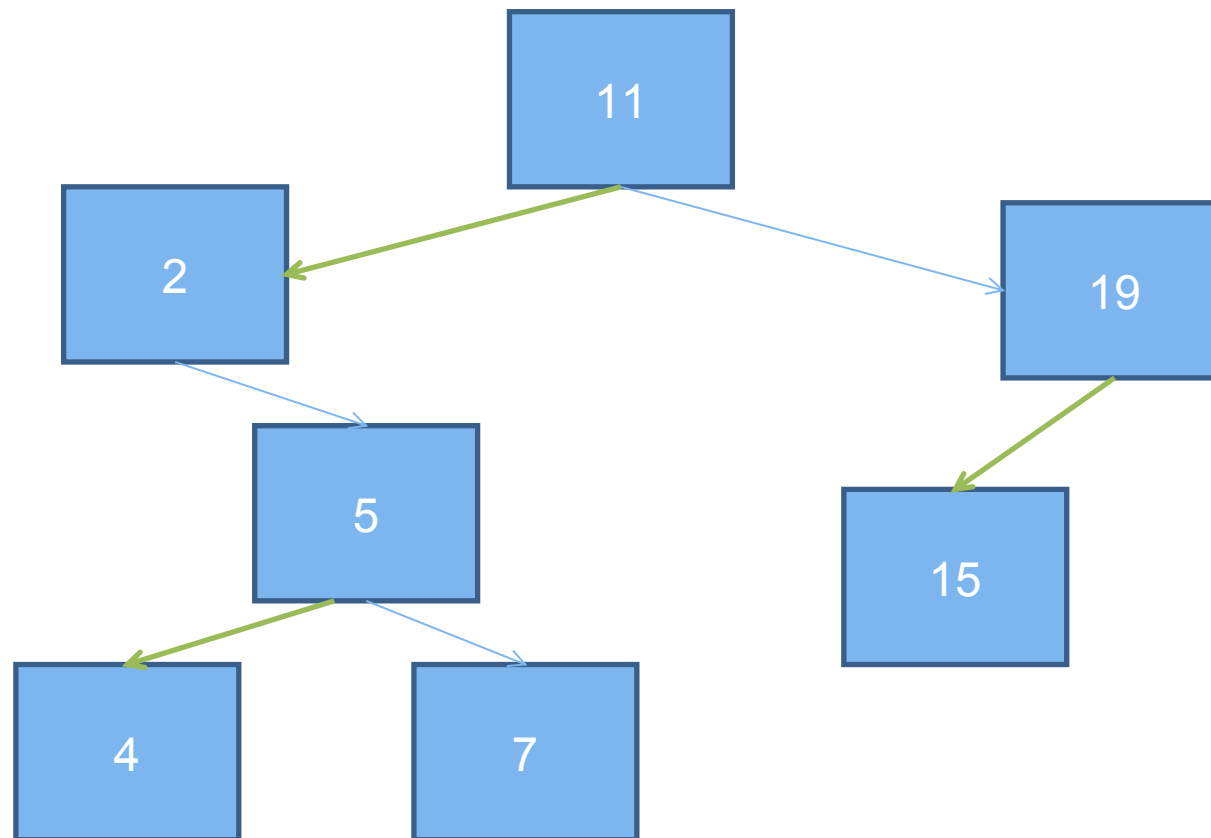
Κατά την κατασκευή του δένδρου τοποθετούμε κόμβους σε θέση κατάλληλη.

Το δένδρο μπορεί να επεκταθεί οποιαδήποτε στιγμή



Δυαδικά Δένδρα

- Παράδειγμα: 11, 2, 5, 19, 7, 15, 4
- Κανόνας: Αριστερά οι μικρότεροι, δεξιά οι μεγαλύτεροι



Διαδικά Δένδρα

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

struct treeNode {
    struct treeNode *leftPtr;
    int data;
    struct treeNode *rightPtr;
};

typedef struct treeNode TreeNode;
typedef TreeNode *TreeNodePtr;

void insertNode( TreeNodePtr *, int );
void inOrder( TreeNodePtr );
void preOrder( TreeNodePtr );
void postOrder( TreeNodePtr );
```



Διαδικά Δένδρα

```
int main()
{
    int i, item;
    TreeNodePtr rootPtr = NULL;

    srand( time( NULL ) );

    /* insert random values between 1 and 15 in the tree */
    printf( "The numbers being placed in the tree are:\n" );

    for ( i = 1; i <= 10; i++ ) {
        item = rand() % 15;
        printf( "%3d", item );
        insertNode( &rootPtr, item );
    }

    /* traverse the tree preOrder */
    printf( "\n\nThe preOrder traversal is:\n" );
    preOrder( rootPtr );

    /* traverse the tree inOrder */
    printf( "\n\nThe inOrder traversal is:\n" );
    inOrder( rootPtr );

    /* traverse the tree postOrder */
    printf( "\n\nThe postOrder traversal is:\n" );
    postOrder( rootPtr );

    return 0;
}
```



Τοποθέτηση

- Κενή θέση;
 - Αν ναι, συνέδεσε τον κόμβο και τέλος.
 - Αν όχι,
 - Αν η τιμή που θέλω να τοποθετήσω είναι μικρότερη από την τρέχουσα, τοποθέτησε αριστερά
 - Αν είναι μεγαλύτερη, τοποθέτησε δεξιά
 - Αν είναι ίση, η τιμή υπάρχει στο δένδρο, τέλος.



Διαδικά Δένδρα

```
void insertNode( TreeNodePtr *treePtr, int value )
{
    if ( *treePtr == NULL ) { /* *treePtr is NULL */
        *treePtr = malloc( sizeof(TreeNode) );

        if ( *treePtr != NULL ) {
            ( *treePtr )->data = value;
            ( *treePtr )->leftPtr = NULL;
            ( *treePtr )->rightPtr = NULL;
        }
        else
            printf( "%d not inserted. No memory available.\n",
                value );
    }
    else
        if ( value < ( *treePtr )->data )
            insertNode( &( ( *treePtr )->leftPtr ), value );
        else if ( value > ( *treePtr )->data )
            insertNode( &( ( *treePtr )->rightPtr ), value );
        else
            printf( "dup" );
}
```



Δυαδικά Δένδρα

```
void inOrder( TreeNodePtr treePtr )
{
    if ( treePtr != NULL ) {
        inOrder( treePtr->leftPtr );
        printf( "%3d", treePtr->data );
        inOrder( treePtr->rightPtr );
    }
}
```



Δυαδικά Δένδρα

```
void preOrder( TreeNodePtr treePtr )
{
    if ( treePtr != NULL ) {
        printf( "%3d", treePtr->data );
        preOrder( treePtr->leftPtr );
        preOrder( treePtr->rightPtr );
    }
}
```

```
void postOrder( TreeNodePtr treePtr )
{
    if ( treePtr != NULL ) {
        postOrder( treePtr->leftPtr );
        postOrder( treePtr->rightPtr );
        printf( "%3d", treePtr->data );
    }
}
```



Σημείωμα αναφοράς

- Copyright Πανεπιστήμιο Πατρών,
Παλιουράς Βασίλειος , Δερματάς Ευάγγελος
«Αρχές Προγραμματισμού ».
Έκδοση: 1.0. Πάτρα 2015
- Διαθέσιμο από τη δικτυακική διεύθυνση
<https://eclass.upatras.gr/modules/>

